Universidad Diego Portales
Escuela de Informática &
Telecomunicaciones

ESTRUCTURAS DE DATOS &
ANÁLISIS DE ALGORITMOS

Laboratorio de algoritmos sobre arreglos

Autores: Kevin Bello
Maillén Andrade

Profesor:
Cristián Llull

27 de Agosto de 2025

Índice

1. Introducción **2**

2. Implementación **2**
2.1. Atributos de la Clase AnalizadorDeNotas 2
2.2. Métodos de la Clase **AnalizadorDeNotas** 3
2.2.1. Codigo final completo: 8

3. Experimentación y Resultados. **12**

4. Conclusión **19**

5. Anexos **20**

1. Introducción

El presente informe detalla el proceso de diseño e implementación de un sistema que permita calcular estadistícas complejas de manera optimizada, el cual debe procesar una gran cantidad de calificaciones, esto lo hara manejando una matriz de calificaciones de tamaño variable, con una fila que representa un estudiante y cada columna una evaluación incluyendo además el manejo de identificadores de estudiantes y los nombres de las evaluaciones

2. Implementación

2.1. Atributos de la Clase AnalizadorDeNotas

Esta clase debera contener los siguientes atributos:

- **double[][] notas**: Una matriz bidimensional que almacena las calificaciones, donde notas[i][j] representa la calificación del estudiante i en la evaluación j.
- **String[] evaluaciones**: Un arreglo con los "títulos" de las evaluaciones. Su tamaño debe coincidir con la cantidad de notas.
- **int[] rut:** arreglo de identificadores asociados a cada uno de los estudiantes (únicos).
- **int cantEstudiantes**: número total de estudiantes (cantidad de filas).
- **int cantEvaluaciones:** número total de evaluaciones (cantidad de columnas).

```
1 import java.util.Random;
2
3 public class AnalizadorDeNotas {
4     private double[][] notas;
5     private String[] evaluaciones;
6     private int[] rut;
7     private int cantEstudiantes;
8     private int cantEvaluaciones;
```

Código 1: Implementacion de atributos

2.2. Métodos de la Clase AnalizadorDeNotas

El objetivo es implementar la clase AnalizadorDeNotas en Java que permita procesar una matriz de calificaciones de gran tamaño y calcular estadísticas para estudiantes y evaluaciones, para lo cual usaremos los siguientes métodos:

- **public AnalizadorDeNotas(int estudiantes, int evaluaciones)**: constructor que recibe las dimensiones de la matriz y genera datos aleatorios. Asigna nombres simples a las evaluaciones para atributo evaluaciones.

```
1     public AnalizadorDeNotas(int estudiantes, int
         evaluaciones) {
2               this.cantEstudiantes = estudiantes;
3               this.cantEvaluaciones = evaluaciones;
4               this.notas = new double[cantEstudiantes][
                   cantEvaluaciones];
5               this.rut = new int[cantEstudiantes];
6               this.evaluaciones = new String[evaluaciones];
7
8               Random random = new Random();
9               for (int i = 0; i < evaluaciones; i++) {
10                  for (int j = 0; j < cantEvaluaciones; j
                       ++) {
11                   this.notas[i][j] = random.nextDouble
                         ()*7.0;
12                  }
13              }
14              for (int j = 0; j < cantEvaluaciones; j++) {
15              this.evaluaciones[j] = "Evaluacion " + (j +
                   1);
16              }
17          for (int i = 0; i < cantEstudiantes; i++) {
18              this.rut[i] = 10000000 + i;
19          }
20    }
21 }
```

Código 2: Constructor de AnalizadorDeNotas

- **public AnalizadorDeNotas(int estudiantes, int evaluaciones, String[] nombresEvaluaciones)**: Constructor que recibe las dimensiones de la matriz y genera datos aleatorios.

```
public AnalizadorDeNotas(int estudiantes, int
   evaluaciones, String[] nombresEvaluaciones) {
        this(estudiantes, evaluaciones); //reutiliza el
           codigo del primer constructor que inicializa
           los objetos
        if (nombresEvaluaciones.length == evaluaciones) {
            this.evaluaciones = nombresEvaluaciones; //
               si el tama o es el mismo, se reemplazan
               los valores iniciales por los nuevos
               nombres de las evaluaciones.
        }
    }
    private void generarNotasAleatorias() {
        Random rand = new Random();
        for (int i = 0; i < cantEstudiantes; i++) {
            for (int j = 0; j < cantEvaluaciones; j++) {
                notas[i][j] = 1.0 + (rand.nextDouble() *
                   6.0); // rango [1.0, 7.0]
            }
        }
    }
```

Código 3: Constructor de AnalizadorDeNotas

- **public double calcularPromedioEstudiante(id numEstudiante):** calcula el promedio de un estudiante basado en el número de estudiante.

```
public double calcularPromedioEstudiante(int
   numEstudiante) {
        double suma = 0;
        for (int j = 0; j < cantEvaluaciones; j++) {
            suma += notas[numEstudiante][j]; //suma las
               notas de la posicion en la fila
               numEstudiante y la columna j.
        }
        return suma / cantEvaluaciones;
    }
```

Código 4: Constructor de AnalizadorDeNotas

- **public double calcularPromedioEvaluacion(int index)**: Calcula el promedio de una evaluación específica.

```
1       public double calcularPromedioEvaluacion(int index)
            {
2           double suma = 0;
3           for (int i = 0; i < cantEstudiantes; i++) {
4               suma += notas[i][index]; //hace la suma por
                    cada estudiante en la posicion i de su
                    nota en la posicion index de la columna de
                     evaluaciones.
5           }
6           return suma / cantEstudiantes;
7       }
```

Código 5: Constructor de AnalizadorDeNotas

- **public double calcularVarianzaEvaluacion(int index)**: calcula la varianza de una evaluación específica.

```
1 public double calcularVarianzaEvaluacion(int index) {
2           double promedio = calcularPromedioEvaluacion(
               index);
3           double suma = 0;
4           for (int i = 0; i < cantEstudiantes; i++) {
5               suma += Math.pow(notas[i][index] - promedio,
                    2);
6           }
7           return suma / cantEstudiantes;
8       }
```

Código 6: Constructor de AnalizadorDeNotas

- **public double[] calcularPromediosEstudiantes():** calcula el promedio de notas de cada estudiante.

```
public double[] calcularPromediosEstudiantes() {
        double[] promedios = new double[cantEstudiantes];
        for (int i = 0; i < cantEstudiantes; i++) {
            promedios[i] = calcularPromedioEstudiante(i);
                //llama a la funcion
                calcularPromedioEstudiante(en posicion i)
                y va guardando en el arreglo de tamano [
                cantEstudiantes].
        }
        return promedios;
    }
```

Código 7: Constructor de AnalizadorDeNotas

- **public double[] calcularVarianzaEstudiantes()**: calcula la varianza de notas de cada estudiante.

```
     public double[] calcularVarianzaEstudiantes() {
        double[] varianzas = new double[cantEstudiantes];
        for (int i = 0; i < cantEstudiantes; i++) {
            double promedio = calcularPromedioEstudiante(
               i);
            double suma = 0;
            for (int j = 0; j < cantEvaluaciones; j++) {
                suma += Math.pow(notas[i][j] - promedio,
                   2);
            }
            varianzas[i] = suma / cantEvaluaciones;
        }
        return varianzas;
    }
```

Código 8: Constructor de AnalizadorDeNotas

- **public double[]calcularPromedioEvaluaciones(String[] evaluaciones):** calcular el promedio de las evaluaciones especificadas, para cada estudiante. Este método retorna un arreglo de tamaño cantEstudiantes.

```
 public double[] calcularPromedioEvaluaciones(String[]
    evals) {
        double[] promedios = new double[cantEstudiantes];
        for (int i = 0; i < cantEstudiantes; i++) {
```

```
            double suma = 0;
            int count = 0;
            for (String eval : evals) { // eval es una
               evaluacion que pasa por el metodo "evals".
                for (int j = 0; j < cantEvaluaciones; j
                   ++) {
                    if (evaluaciones[j].equals(eval)) {
                       // equals() compara el contenido
                       del objeto, a diferencia del == .
                       La linea revisa si la evaluacion
                       de la matriz coincide con el
                       solicitado por el usuario.
                        suma += notas[i][j];
                        count++;
                    }
                }
            }
            if (count > 0) {
                promedios[i] = suma / count;
            } else {
                promedios[i] = 0;
            }
        }
        return promedios;
    }
```

Código 9: Constructor de AnalizadorDeNotas

- **public String encontrarMaximo(int index)**: determina la nota máxima para la evaluación seleccionada, retornando el identificador del estudiante que la obtuvo (si dos o más estudiantes obtienen el máximo, retornar el primero).

```
    public String encontrarMaximo(int index) {
        double maxNota = notas[0][index];
        int pos = 0;
        for (int i = 1; i < cantEstudiantes; i++) {
            if (notas[i][index] > maxNota) {
                maxNota = notas[i][index];
                pos = i;
            }
        }
        return "Rut: " + rut[pos] + " | Nota: " + maxNota
           ;
    }
```

Código 10: Constructor de AnalizadorDeNotas

2.2.1. Codigo final completo:

```
import java.util.Random;
import java.lang.Math;

public class AnalizadorDeNotas{

    private double[][] notas;
    private String[] evaluaciones;
    private int[] rut;
    private int cantEstudiantes;
    private int cantEvaluaciones;
    private double [] promediosEstudiantes;
    private double [] promediosEvaluaciones;

    public AnalizadorDeNotas(int estudiantes, int
       evaluaciones) {
        this.cantEstudiantes = estudiantes;
        this.cantEvaluaciones = evaluaciones;
        this.notas = new double[estudiantes][evaluaciones];
        this.rut = new int[estudiantes];
        this.evaluaciones = new String[evaluaciones];

        for (int i = 0; i < estudiantes; i++) {
            rut[i] = 1000 + i;
        }

        for (int j = 0; j < evaluaciones; j++) {
            this.evaluaciones[j] = "Evaluacion" + (j + 1);
        }

        generarNotasAleatorias();
        calcularYAlmacenarPromedios();
    }

    public AnalizadorDeNotas(int estudiantes, int
       evaluaciones, String[] nombresEvaluaciones) {
        this(estudiantes, evaluaciones);
        if (nombresEvaluaciones.length == evaluaciones) {
            this.evaluaciones = nombresEvaluaciones;
        }
    }

    private void generarNotasAleatorias() {
        Random rand = new Random();
        for (int i = 0; i < cantEstudiantes; i++) {
```

```
43              for (int j = 0; j < cantEvaluaciones; j++) {
44                  notas[i][j] = 1.0 + (rand.nextDouble() * 6.0)
                        ;
45              }
46          }
47      }
48
49      public double calcularPromedioEstudiante(int
            numEstudiante) {
50          double suma = 0;
51          for (int j = 0; j < cantEvaluaciones; j++) {
52              suma += notas[numEstudiante][j];
53          }
54          return suma / cantEvaluaciones;
55      }
56      private void calcularYAlmacenarPromedios() {
57          this.promediosEstudiantes = new double[
                cantEstudiantes];
58          this.promediosEvaluaciones = new double[
                cantEvaluaciones];
59
60          for(int i = 0; i < cantEstudiantes; i++) {
61              promediosEstudiantes[i] =
                    calcularPromedioEstudiante(i);
62          }
63          for(int j = 0; j < cantEvaluaciones; j++) {
64              promediosEvaluaciones[j] =
                    calcularPromedioEvaluacion(j);
65          }
66      }
67
68      public double[] calcularPromediosEstudiantesOptimizado()
            {
69          return promediosEstudiantes;
70      }
71
72      public double[] calcularPromedioEvaluacionesOptimizado()
            {
73          return promediosEvaluaciones;
74      }
75
76      public double calcularPromedioEvaluacion(int index) {
77          double suma = 0;
78          for (int i = 0; i < cantEstudiantes; i++) {
79              suma += notas[i][index];
80          }
```

```
81          return suma / cantEstudiantes;
82      }
83
84      public double calcularVarianzaEvaluacion(int index) {
85          double promedio = calcularPromedioEvaluacion(index);
86          double suma = 0;
87          for (int i = 0; i < cantEstudiantes; i++) {
88              suma += Math.pow(notas[i][index] - promedio, 2);
89          }
90          return suma / cantEstudiantes;
91      }
92
93      public double[] calcularPromediosEstudiantes() {
94          double[] promedios = new double[cantEstudiantes];
95          for (int i = 0; i < cantEstudiantes; i++) {
96              promedios[i] = calcularPromedioEstudiante(i);
97          }
98          return promedios;
99      }
100
101     public double[] calcularVarianzaEstudiantes() {
102         double[] varianzas = new double[cantEstudiantes];
103         for (int i = 0; i < cantEstudiantes; i++) {
104             double promedio = calcularPromedioEstudiante(i);
105             double suma = 0;
106             for (int j = 0; j < cantEvaluaciones; j++) {
107                 suma += Math.pow(notas[i][j] - promedio, 2);
108             }
109             varianzas[i] = suma / cantEvaluaciones;
110         }
111         return varianzas;
112     }
113
114     public double[] calcularPromedioEvaluaciones(String[]
        evals) {
115         double[] promedios = new double[cantEstudiantes];
116         for (int i = 0; i < cantEstudiantes; i++) {
117             double suma = 0;
118             int count = 0;
119             for (String eval : evals) {
120                 for (int j = 0; j < cantEvaluaciones; j++) {
121                     if (evaluaciones[j].equals(eval)) {
122                         suma += notas[i][j];
123                         count++;
124                     }
125                 }
```

```
126             }
127             if (count > 0) {
128                 promedios[i] = suma / count;
129             } else {
130                 promedios[i] = 0;
131             }
132         }
133         return promedios;
134     }
135 
136     public String encontrarMaximo(int index) {
137         double maxNota = notas[0][index];
138         int pos = 0;
139         for (int i = 1; i < cantEstudiantes; i++) {
140             if (notas[i][index] > maxNota) {
141                 maxNota = notas[i][index];
142                 pos = i;
143             }
144         }
145         return "Rut: " + rut[pos] + " | Nota: " + maxNota;
146     }
147 }
```

Código 11: Codigo final

3. Experimentación y Resultados.

En esta sección se verifica que los algoritmos funcionen correctamente, comparando su eficiencia en términos de velocidad. Para ésto mediremos los tiempos de ejecución en nanosegundos utilizando el método **System.nanoTime()**.

Ademas se configuró la matriz de calificaciones para cada prueba. Se creo con un número fijo de **10 evaluaciones** y un número de estudiantes que varía desde **100** hasta **10,000**, en incrementos de **100**.

La función main del archivo ***Experimentacion.java*** se modificó para automatizar las pruebas. El experimento consistió en variar el **número de estudiantes en incrementos** de **100**, desde **100** hasta **10.000**, mientras se mantenía un número fijo de **10 evaluaciones** para cada prueba. Para la generación de los datos, las notas de la matriz se asignaron de forma aleatoria.

```
1  public class Experimentacion {
2      public static void main(String[] args) {
3          int cantEvaluacionesFijas = 10;
4
5          // Para la tabla
6          System.out.println("Estudiantes | T. Prom. | T. Prom.
               Opt. | T. Eval. | T. Eval. Opt. | T. Varianza | T.
               M ximo");
7
8          for (int i = 1; i <= 100; i++) {
9              int cantEstudiantes = 100 * i;
10
11             // Instancia de la clase AnalizadorDeNotas para
                   cada tama o de matriz
12             AnalizadorDeNotas analizador = new
                   AnalizadorDeNotas(cantEstudiantes,
                   cantEvaluacionesFijas);
13
14             // Mide el tiempo de calcularPromediosEstudiantes
                   ()
15             long tiempoInicio1 = System.nanoTime();
16             analizador.calcularPromediosEstudiantes();
17             long tiempoFin1 = System.nanoTime();
18             long duracion1 = tiempoFin1 - tiempoInicio1;
19
20             // MMide el tiempo de
                   calcularPromediosEstudiantesOptimizado()
21             long tiempoInicio2 = System.nanoTime();
22             analizador.calcularPromediosEstudiantesOptimizado
                   ();
```

```
23              long tiempoFin2 = System.nanoTime();
24              long duracion2 = tiempoFin2 - tiempoInicio2;
25
26              //Mide el tiempo de calcularPromedioEvaluaciones
                    ()
27              long tiempoInicio5 = System.nanoTime();
28              analizador.calcularPromedioEvaluaciones(new
                    String[0]); // El par metro no afecta la
                    medici n del tiempo
29              long tiempoFin5 = System.nanoTime();
30              long duracion5 = tiempoFin5 - tiempoInicio5;
31
32              // Mide el tiempo de
                    calcularPromedioEvaluacionesOptimizado()
33              long tiempoInicio6 = System.nanoTime();
34              analizador.calcularPromedioEvaluacionesOptimizado
                    ();
35              long tiempoFin6 = System.nanoTime();
36              long duracion6 = tiempoFin6 - tiempoInicio6;
37
38              // Mide el tiempo de calcularVarianzaEstudiantes
                    ()
39              long tiempoInicio3 = System.nanoTime();
40              analizador.calcularVarianzaEstudiantes();
41              long tiempoFin3 = System.nanoTime();
42              long duracion3 = tiempoFin3 - tiempoInicio3;
43
44              // Mide el tiempo de encontrarMaximo()
45              long tiempoInicio4 = System.nanoTime();
46              analizador.encontrarMaximo(0);
47              long tiempoFin4 = System.nanoTime();
48              long duracion4 = tiempoFin4 - tiempoInicio4;
49
50              // Imprime los resultados
51              System.out.println(cantEstudiantes + " | " +
                    duracion1 + " | " + duracion2 + " | " +
                    duracion5 + " | " + duracion6 + " | " +
                    duracion3 + " | " + duracion4);
52          }
53      }
54  }
```

Código 12: Main de experimentacion

Los valores del tiempo de ejecucion (en **nanosegundos(*ns*)**) para los metodos de la clase AnalizadorDeNotas son los siguientes, separados por **(Estudiantes/Tiempo promedio Estudiante/Tiempo promedio Estudiante Optimizado/tiempo Promedio Evaluacion/ Tiempo promedio Evaluacion Optimizado/Tiempo Varianza/Tiempo Maximo/)**

Estudiantes	T. Prom.	T. Prom. Opt.	T. Eval.	T. Eval. Opt.	T. Varianza	T. Máximo
100	38900	2000	11500	1300	261200	12124800
200	20400	300	16300	200	111700	63000
300	30000	300	17300	500	177400	104400
400	48000	300	20600	200	308600	64100
500	51500	200	25300	200	341600	79700
600	75000	600	32600	400	397100	67700
700	74600	300	32500	400	658000	108300
800	69400	300	40000	800	455600	81400
900	52400	300	40900	200	615900	77100
1000	59200	200	90700	400	581200	85700
1100	102900	300	49300	200	669300	99100
1200	55600	200	41700	200	801600	77700
1300	77700	500	67700	400	217900	122200
1400	80300	300	57800	200	203900	90700
1500	99900	300	68400	300	198900	94900
1600	98700	200	66400	200	236000	97800
1700	131700	200	79700	300	252600	112000
1800	112100	200	133800	600	265800	145200
1900	112700	400	122700	300	208500	138400
2000	142600	200	61900	100	139200	114300

Figura 1: Tiempos de ejecución de los métodos de la clase **AnalizadorDeNotas** en **nanosegundos (*ns*)**.

Con la recopilacion de estos datos podemos llevarlo a los siguientes graficos:

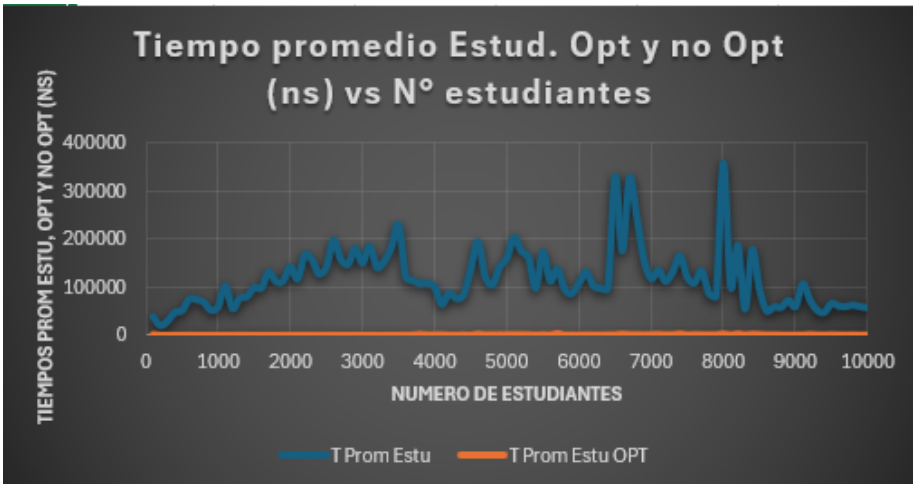

Figura 2: Tiempos promedios de estudiantes Optimizados y no Optimizado en Nanosegundos(ns) en relacion al numero de estudiantes

En el grafico podemos observar que el tiempo de ejecución del algoritmo no optimizado aumenta de manera lineal a medida que incrementa el numero de estudiantes. Esto es debido a que el algoritmo debe recorrer la matriz completa para calcular cada promedio, lo que resulta en una relación directamente proporcional entre tiempo y tamaño de datos **(O(n))**. En cambio la versión optimizada muestra un tiempo de ejecución muy cercano a cero y casi constante, eso es debido a que precalcula y almacena los promedios, por lo que el tiempo de consulta posterior es casi instantáneo **(O(1))**

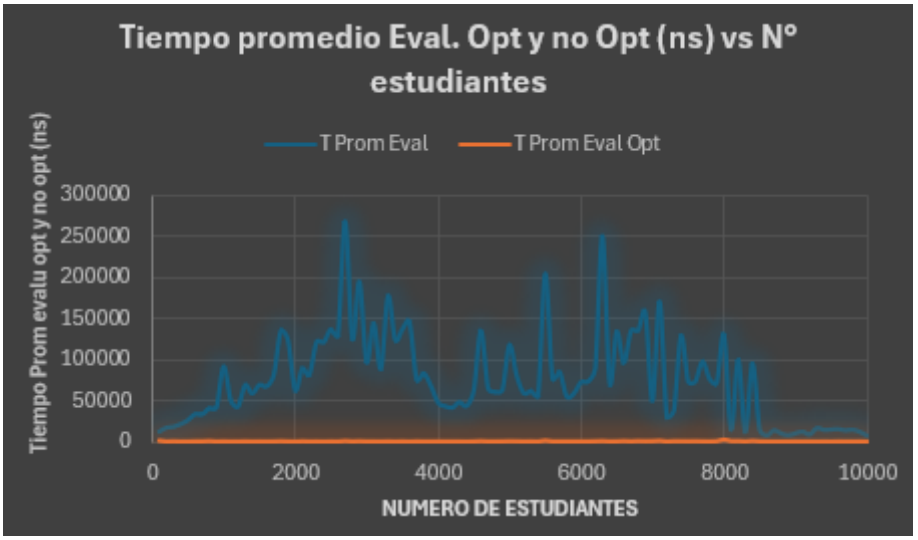

Figura 3: Tiempos promedios de evaluaciones Optimizadas y no Optimizada en Nanosegundos(ns) en relacion al numero de estudiantes

De forma similiar al grafico anterior, el tiempo de ejecución del método no optimizado muestra un crecimiento lineal conforme aumenta el numero de estudiantes, debido que a debe recorrer cada fila de la matriz de notas para calcular el promedio de una evaluación, lo que implica una complejidad de tiempo $\mathbf{O(n)}$. Por otro lado, la versión optimizada demuestra una mejor eficiencia con un tiempo casi constante y muy bajo, de igual manera se logra al almacenar los promedios previamente calculados , esto permite un tiempo constante $\mathbf{O(1)}$

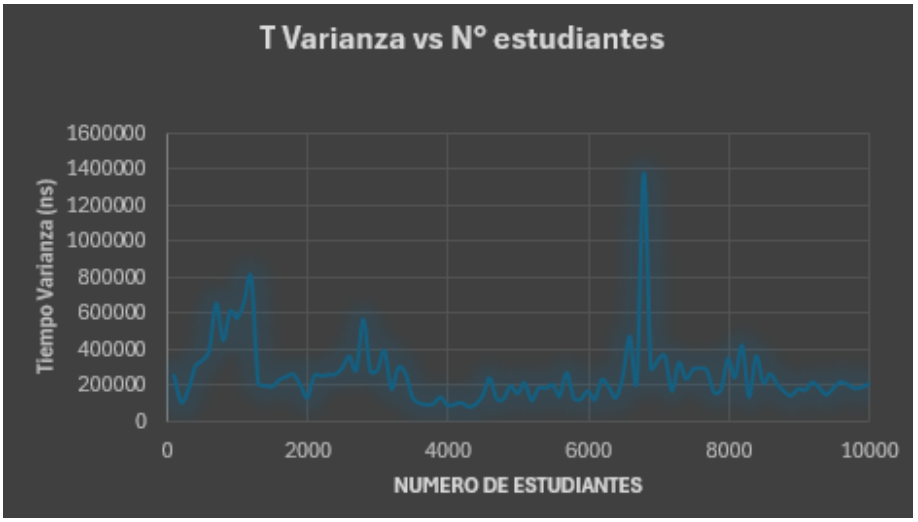

Figura 4: **Tiempo de Varianza** en **Nanosegundos(*ns*)** en relacion vs al **Numero de estudiantes**

En el grafico de varianza muestra un crecimiento cuadratico $(\mathbf{O}(n^2))$ en el tiempo de ejecución a medida que el numero de estudiantes aumenta. Esto sucede porque el método debe recorrer una matriz bidimension (estudiantes y evaluaciones) para realizar el calculo de la varianza, esto significa que por cada estudiante el algoritmo debe iterar a través de todas las evaluaciones, lo que resulta en un tiempo de procesamiento que escala exponencialmente con el tamaño de los datos, lo que nos demuestra que no es eficiente para grandes cantidades de datos

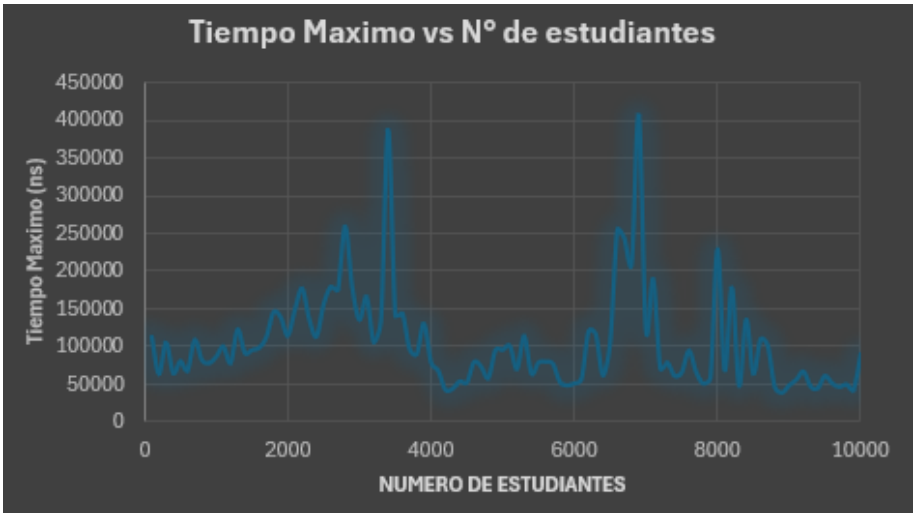

Figura 5: **Tiempo Maximo** en relacion al **Numero de estudiantes**

En este grafico podemos observar que el crecimiento del tiempo de ejecución del método **T.Maximo** tiene un crecimiento lineal **(O(n))** a medida que el numero de estudiantes incrementa, debido a que el algoritmo recorre una columna de la matriz (la de una evaluación especifica) para encontrar el valor mas alto, por lo que el tiempo de ejecución es directamente proporcional al numero de estudiantes que debe procesar

4. Conclusión

Durante el desarrollo del laboratorio, una de las dificultades principales fue el encontrar una manera de optimizar **calcularPromedioEstudiantes()** y **calcularPromedioEvaluaciones()**, ya que debido a errores en sintaxis y demás, se complicaba compilar el código. Esto nos llevo a dar uso al almacenamiento de resultados, en lugar de recalcular los promedios cada vez que se les solicitaba, la clase **AnalizadorDeNotas** fue modificada para almacenar los promedios de estudiantes y evaluaciones en atributos internos de la clase, llamados **promediosEstudiantes** y **promediosEvaluaciones**. Estos promedios se calculan una sola vez al inicializar la clase. De esta forma, cuando se llaman los métodos optimizados, estos no realizan ningún cálculo, sino que simplemente devuelven el valor que ya está guardado.

Al finalizar la experimentación pudimos notar el como varía el rendimiento de los algoritmos en función del tamaño de la matriz

- **Algoritmos no optimizados**: Los métodos ***calcularPromediosEstudiantes()***, ***calcularVarianzaEstudiantes()*** y ***encontrarMaximo()*** muestran un aumento significativo en su tiempo de ejecución a medida que el número de estudiantes aumenta. Esto se debe a que su complejidad es de tipo lineal **(O(n))**, ya que tienen que recorrer la matriz completa para cada cálculo.
- **Algoritmos optimizados**: El método ***calcularPromediosEstudiantesOptimizado()*** presenta un tiempo de ejecución que es casi constante y muy bajo, sin importar el tamaño de la matriz. Su complejidad es de **tipo constante (O(1))**. Esto confirma que la optimización funcionó, ya que el método simplemente accede a un valor previamente calculado en lugar de procesar los datos nuevamente.

5. Anexos

Para acceder al código completo de este proyecto, puede visitar el siguiente enlace: `https://github.com/Kevinb1103/Laboratorio_1/tree/master/src`

Para acceder al template de latex de este proyecto, puede visitar el siguiente enlace: `https://github.com/Kevinb1103/Lab1-EDA/blob/Codigo/Lab_1.zip`